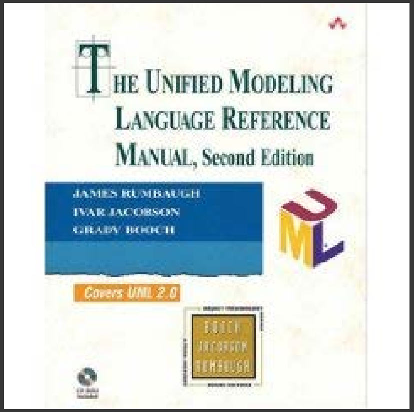
UML SEQUENCE DIAGRAM

Concepts & Notations

Acknowledgements

- The material in this tutorial is based in part on:
  - The Unified Modeling Language Reference Manual, 2nd edition, by James Rumbaugh, Ivar Jacobson, and Grady Booch



The Unified Modeling Language Reference Manual, Second Edition

James Rumbaugh
Ivar Jacobson
Grady Booch

Covers UML 2.0

UML Sequence Diagram

- Interaction View
- Interaction Diagram
- Sequence Diagram
- Examples

Interaction View

- Objects interact to implement behavior

- Two ways to describe interaction
  - focus on interactions of a collection of cooperating objects (interaction view)
  - focus on individual objects (state machine) (we’ll cover this later)

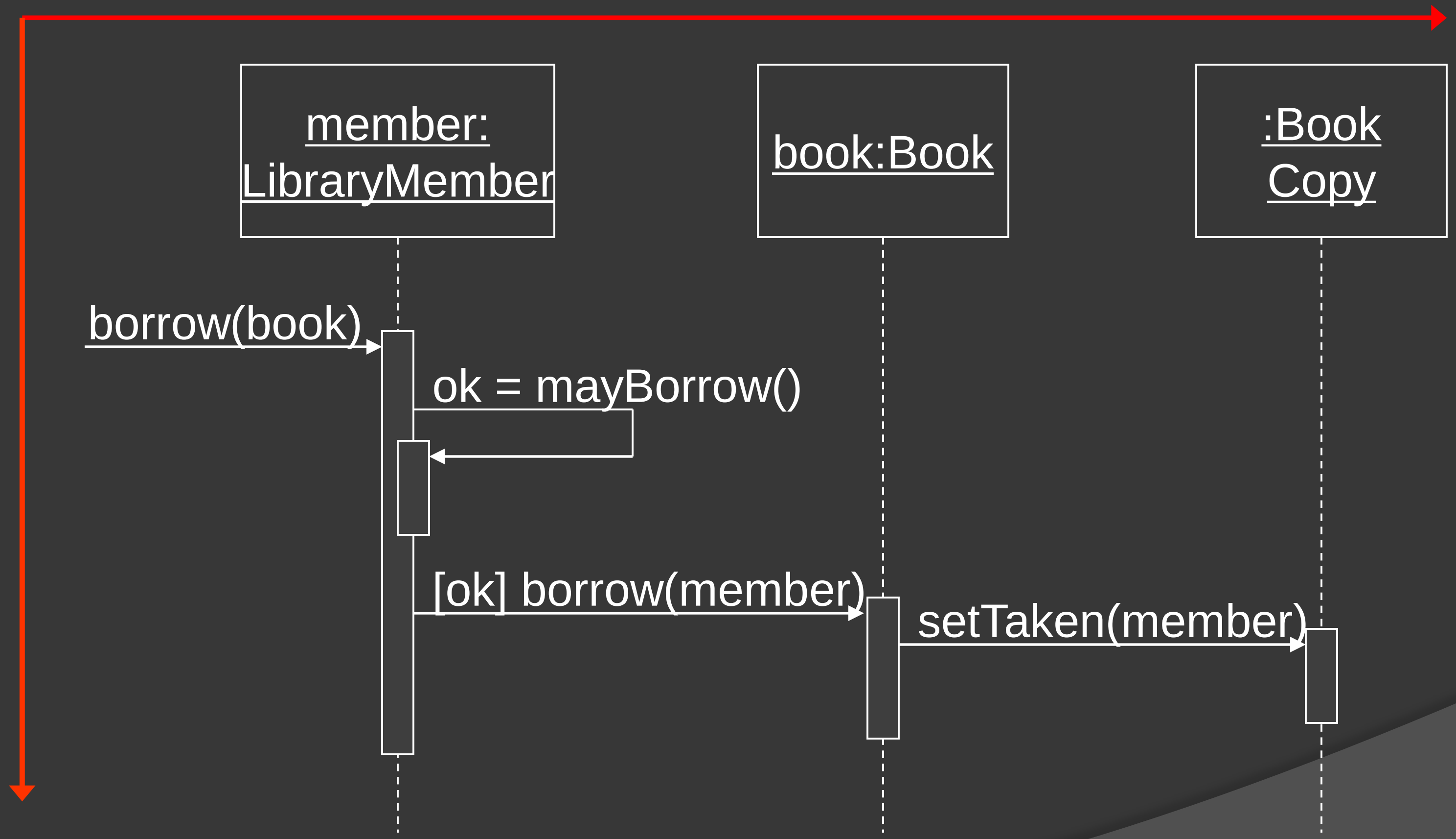
- Provides a more holistic view of the behavior of a set of objects

Interaction Diagram

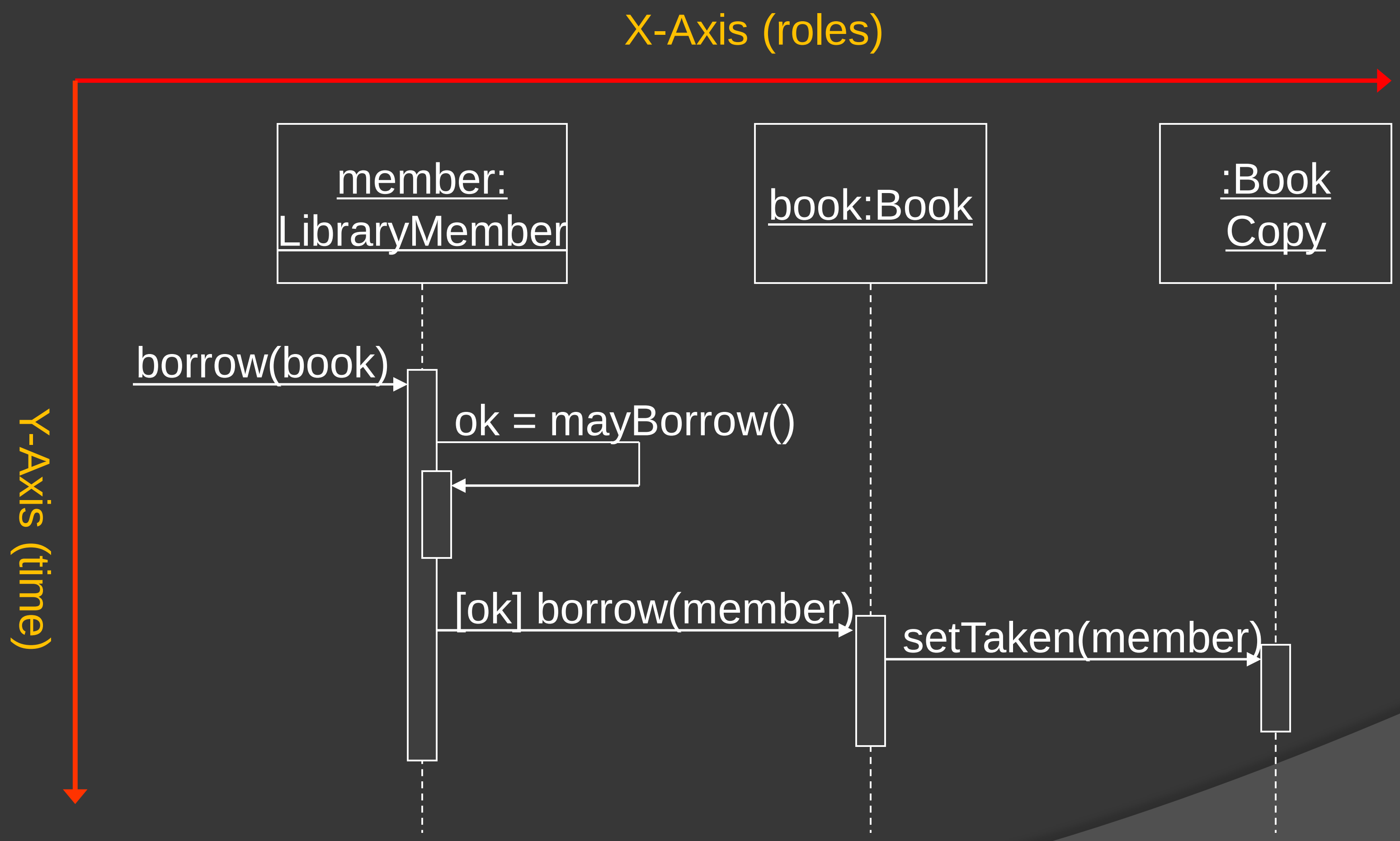
- Collaboration Diagram
  - Emphasizes structural relations between objects

- Sequence Diagram
  - Illustrates how objects interacts with each other
  - Emphasizes time ordering of messages

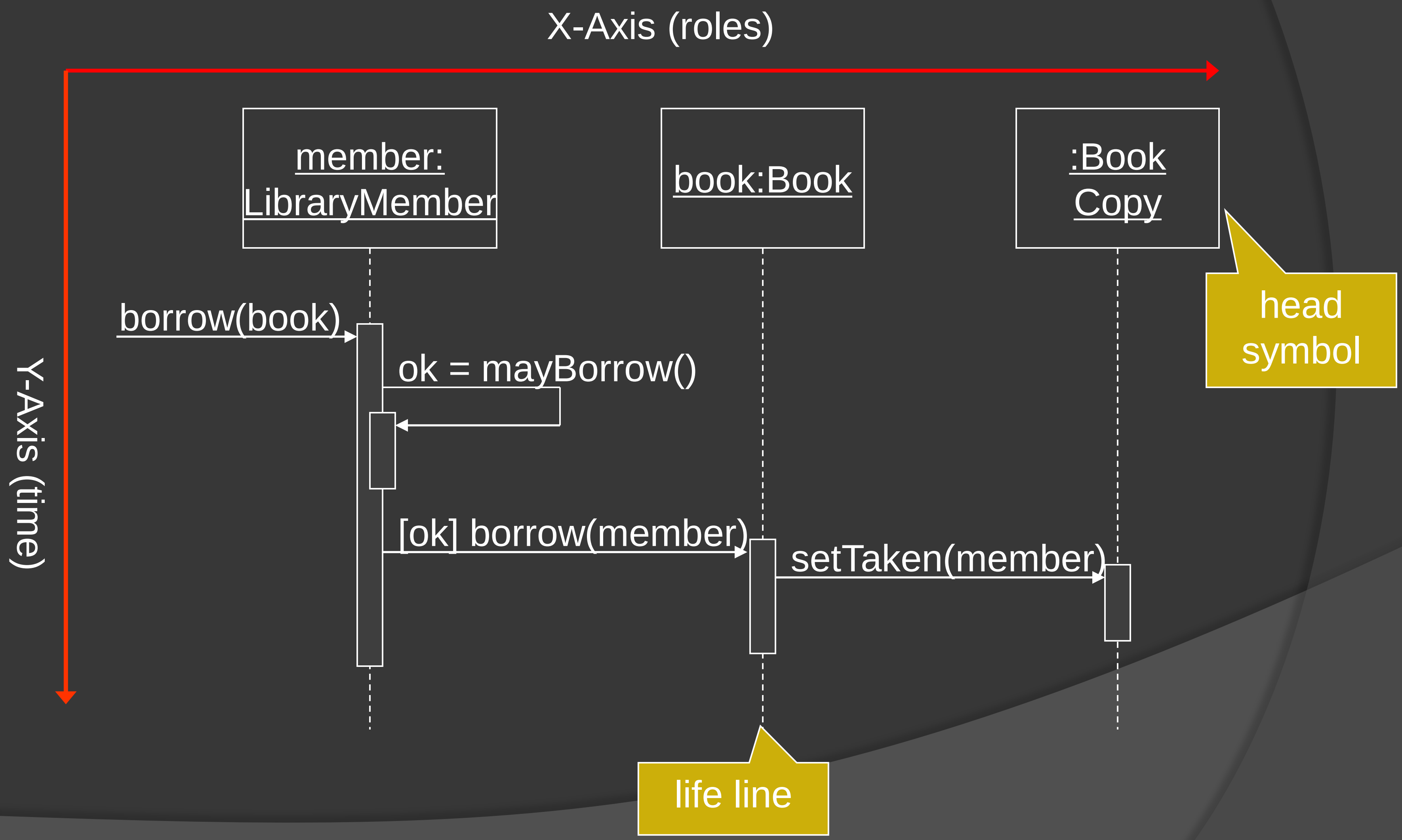
Sequence Diagram



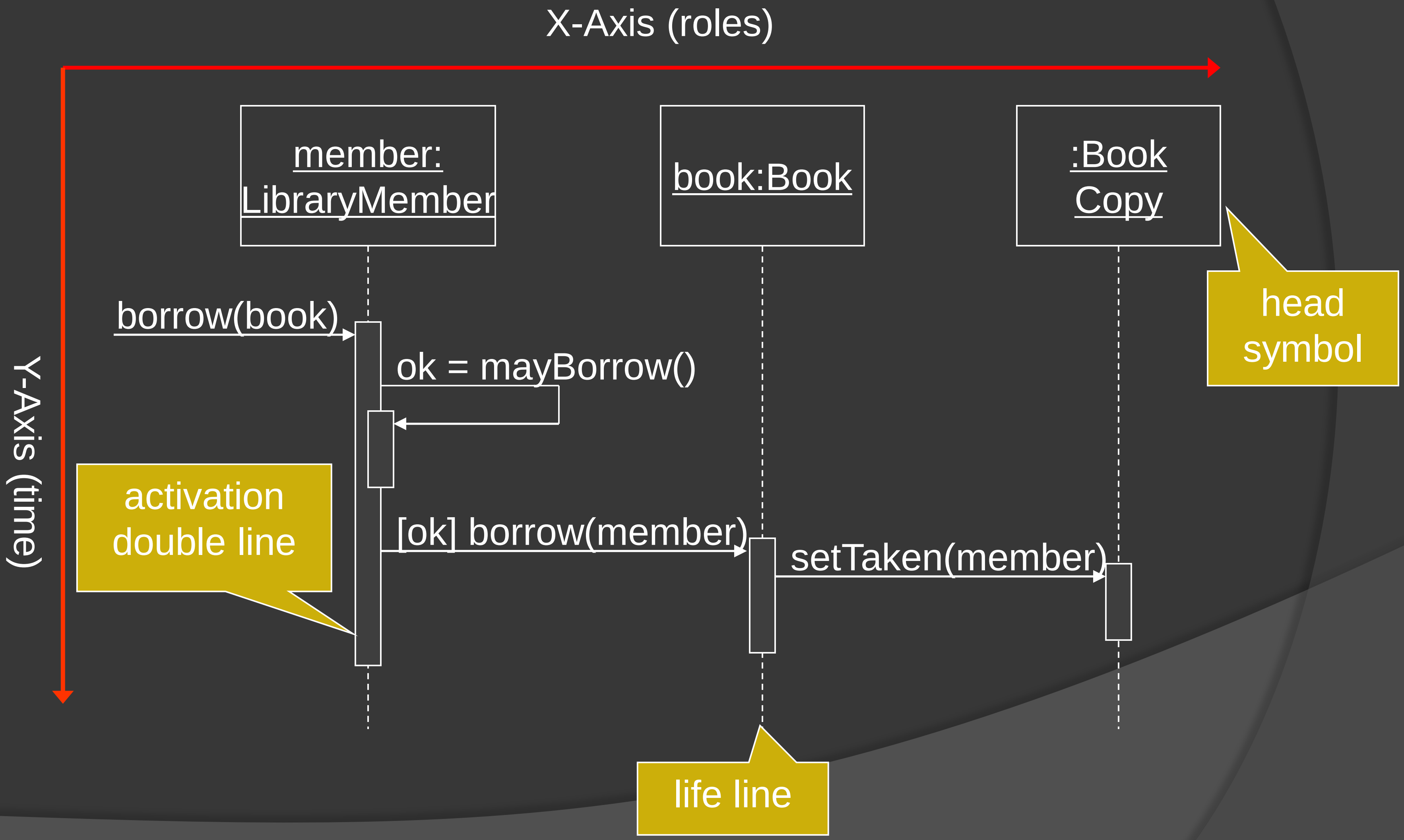
Sequence Diagram



Sequence Diagram



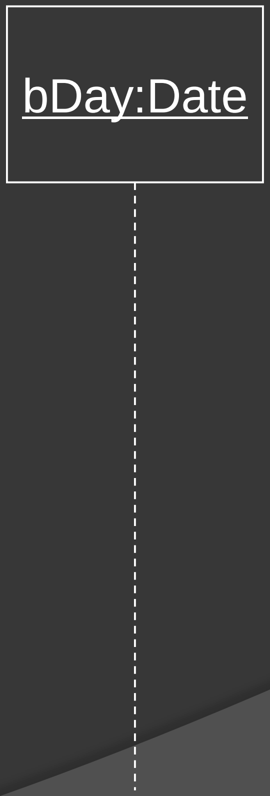
Sequence Diagram



Object

- Naming
  - syntax
  - [instance Name]:[class Name]

- Life line
  - represents the object’s life during the interaction

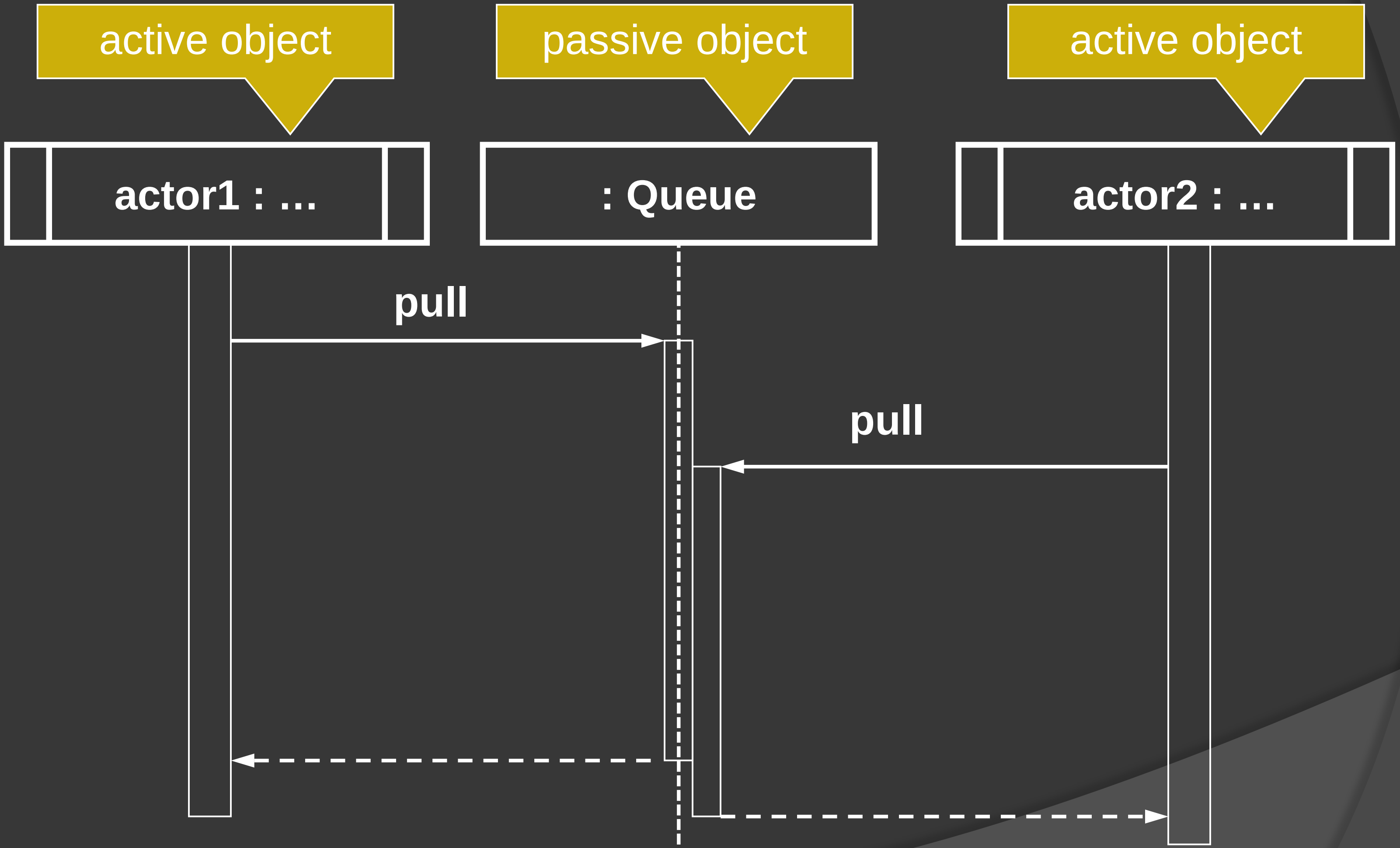


Object

- Active Object
  - holds the root of a stack executions
  - has its own thread of control

- Passive Object
  - objects that are called by an active object
  - receive control only when called

Object


active object
passive object
active object
actor1 : …
: Queue
actor2 : …
pull
pull

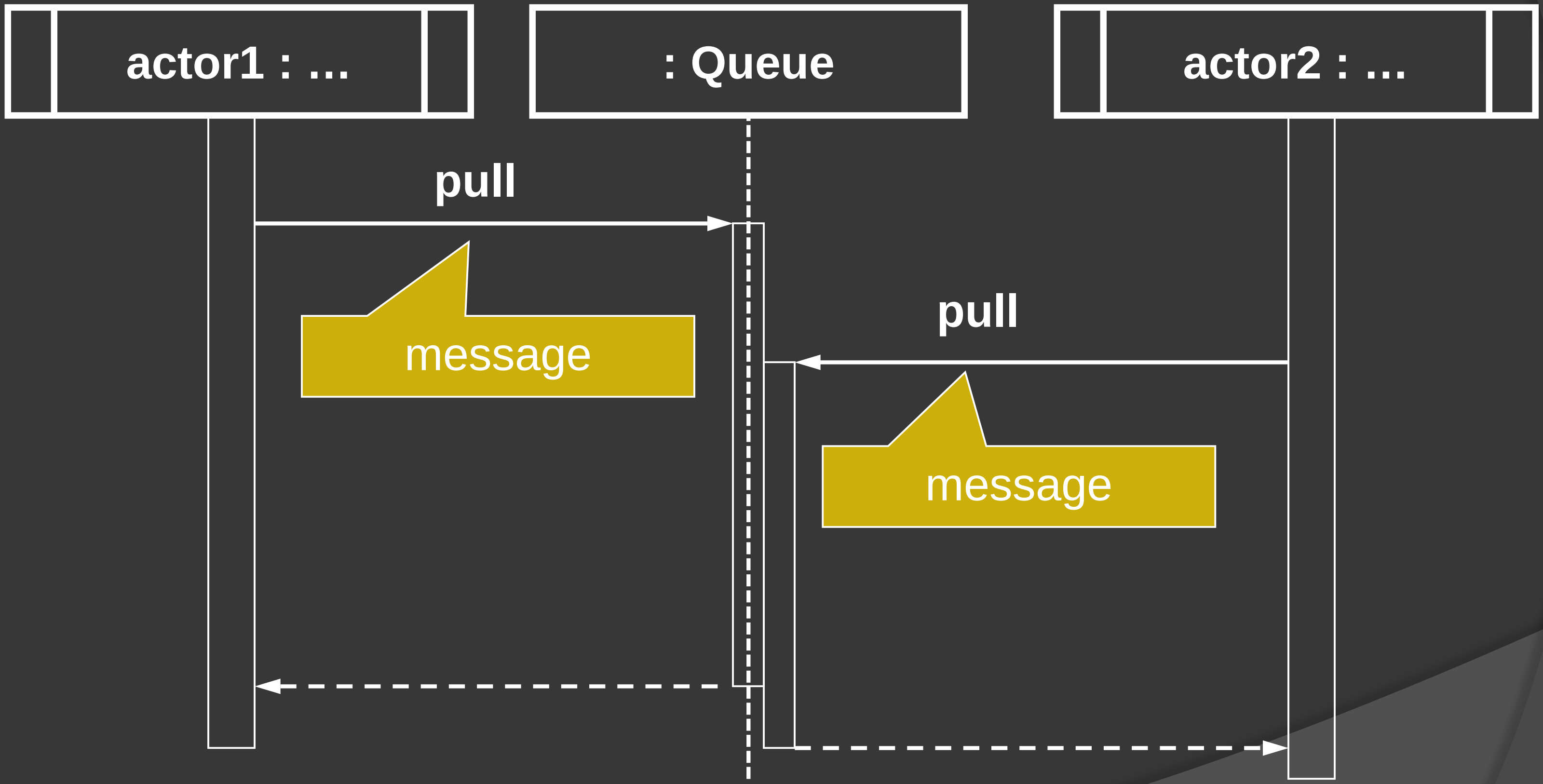
Message

- An interaction between two objects
  - operation call
  - signaling
  - RPC

- An arrow between the life lines of two objects

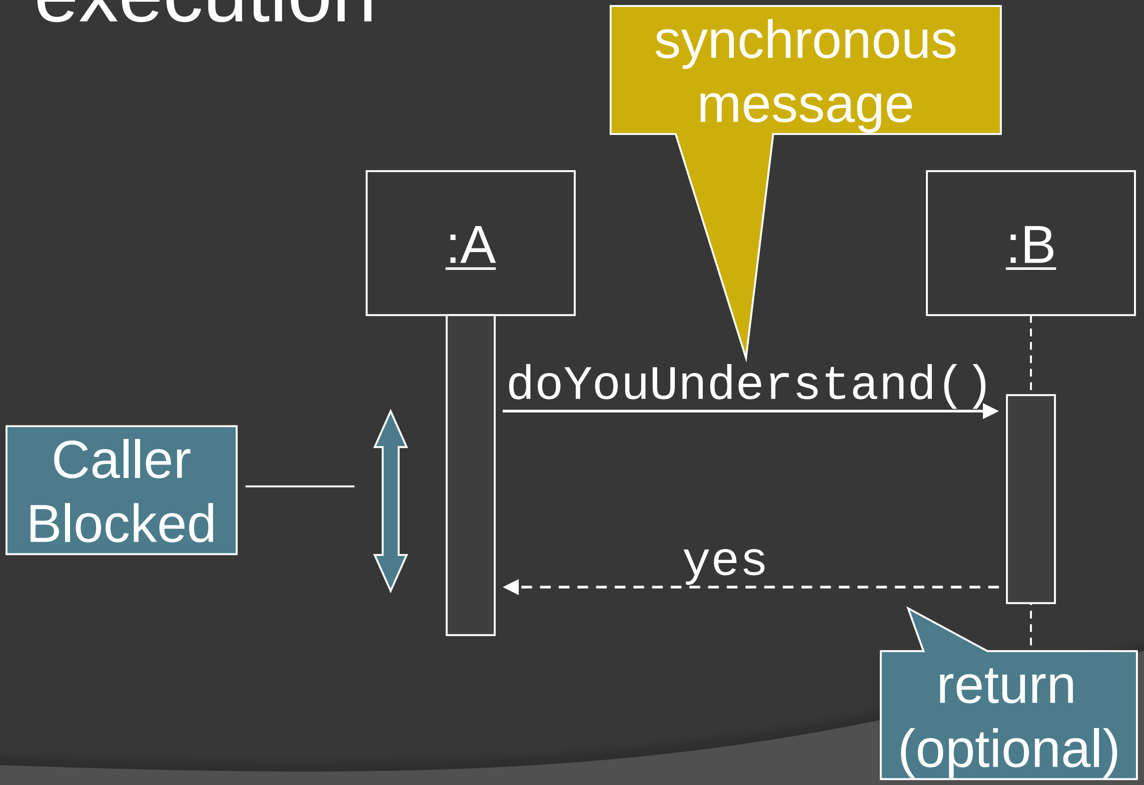
- Labeled with
  - name
  - arguments
  - control information

Message



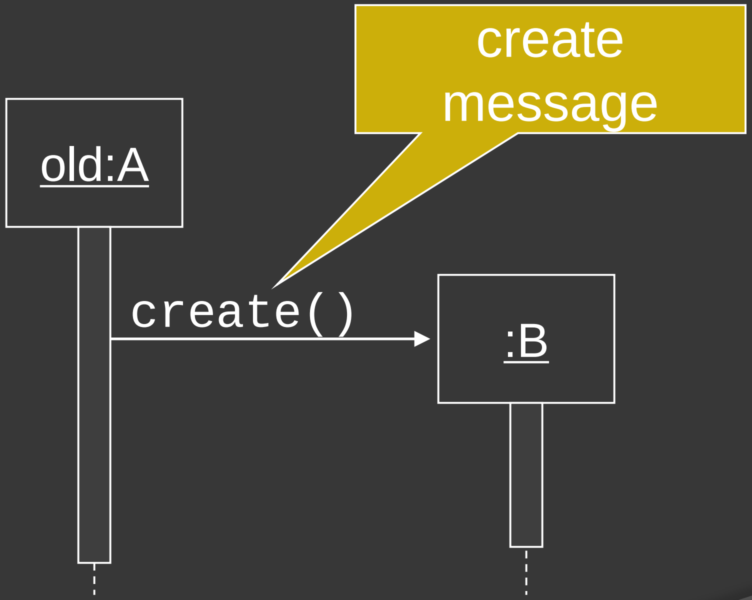
Synchronous Message

- The routine that handles the message is completed before the caller resumes execution



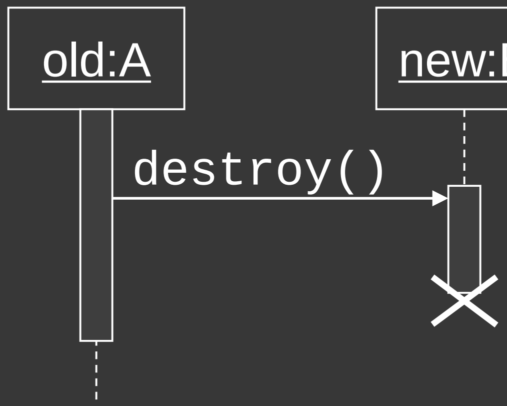
Creation Message

- An object may create another object via a create() message

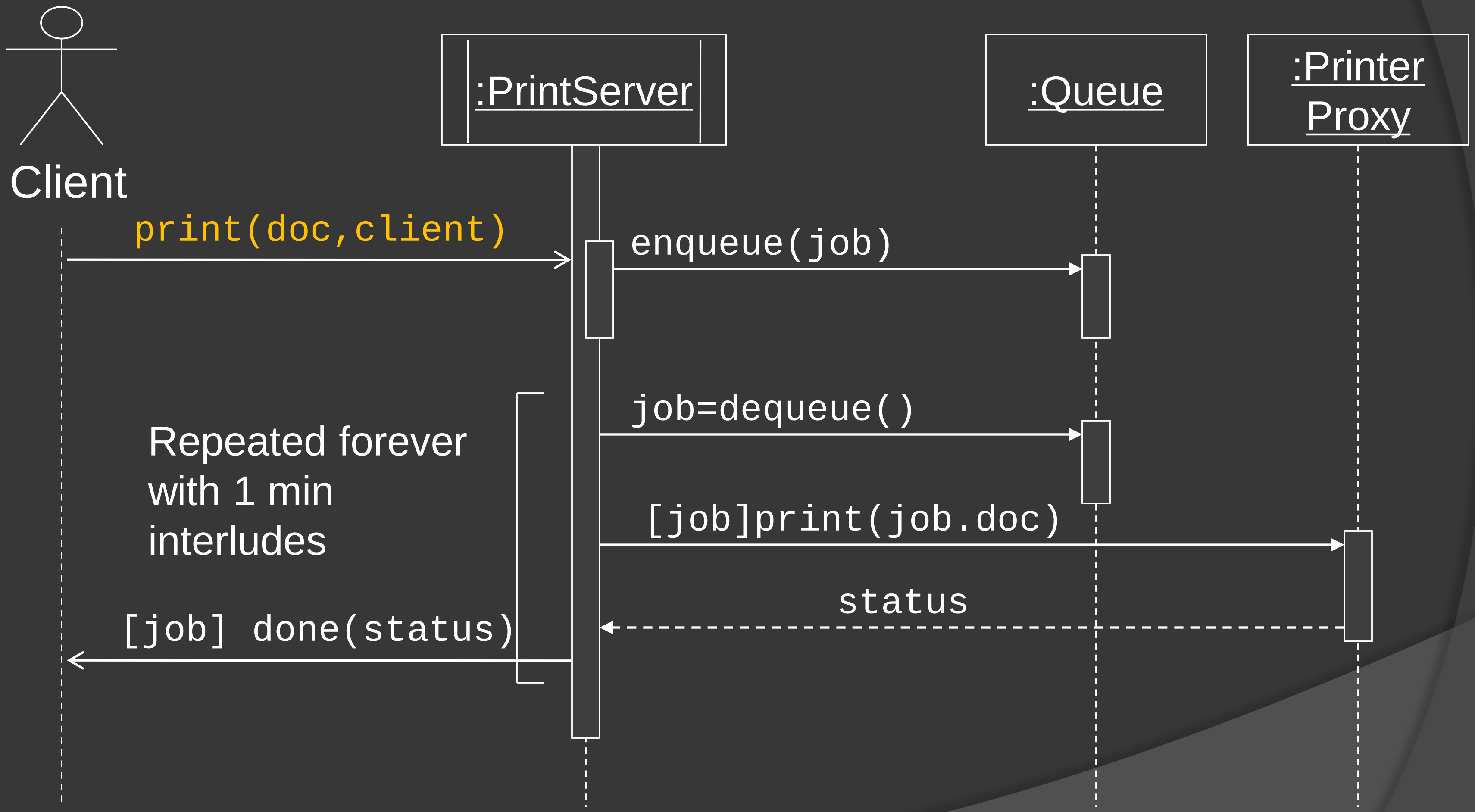


Destruction Message

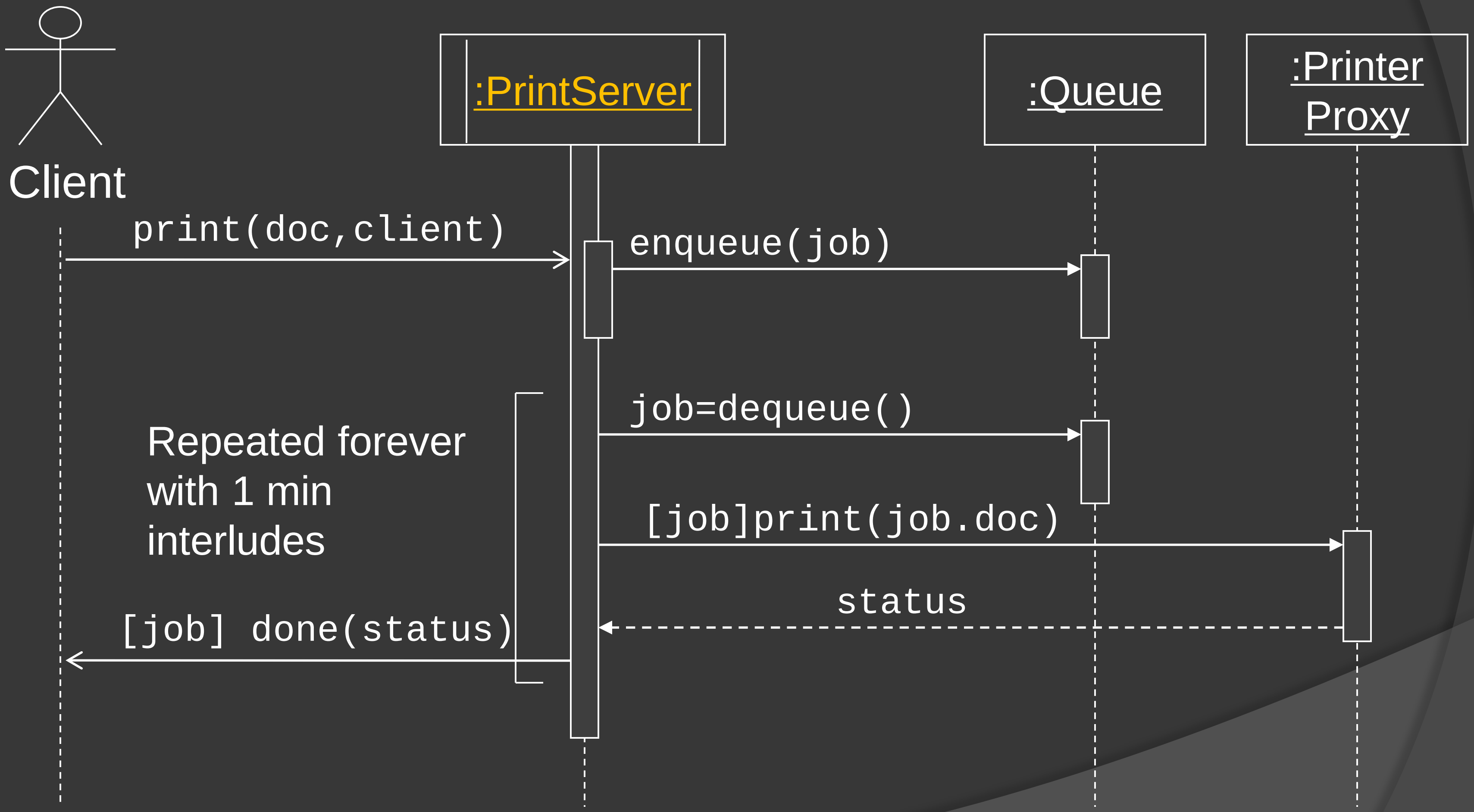
- An object may destroy another object via a destroy() message



Examples

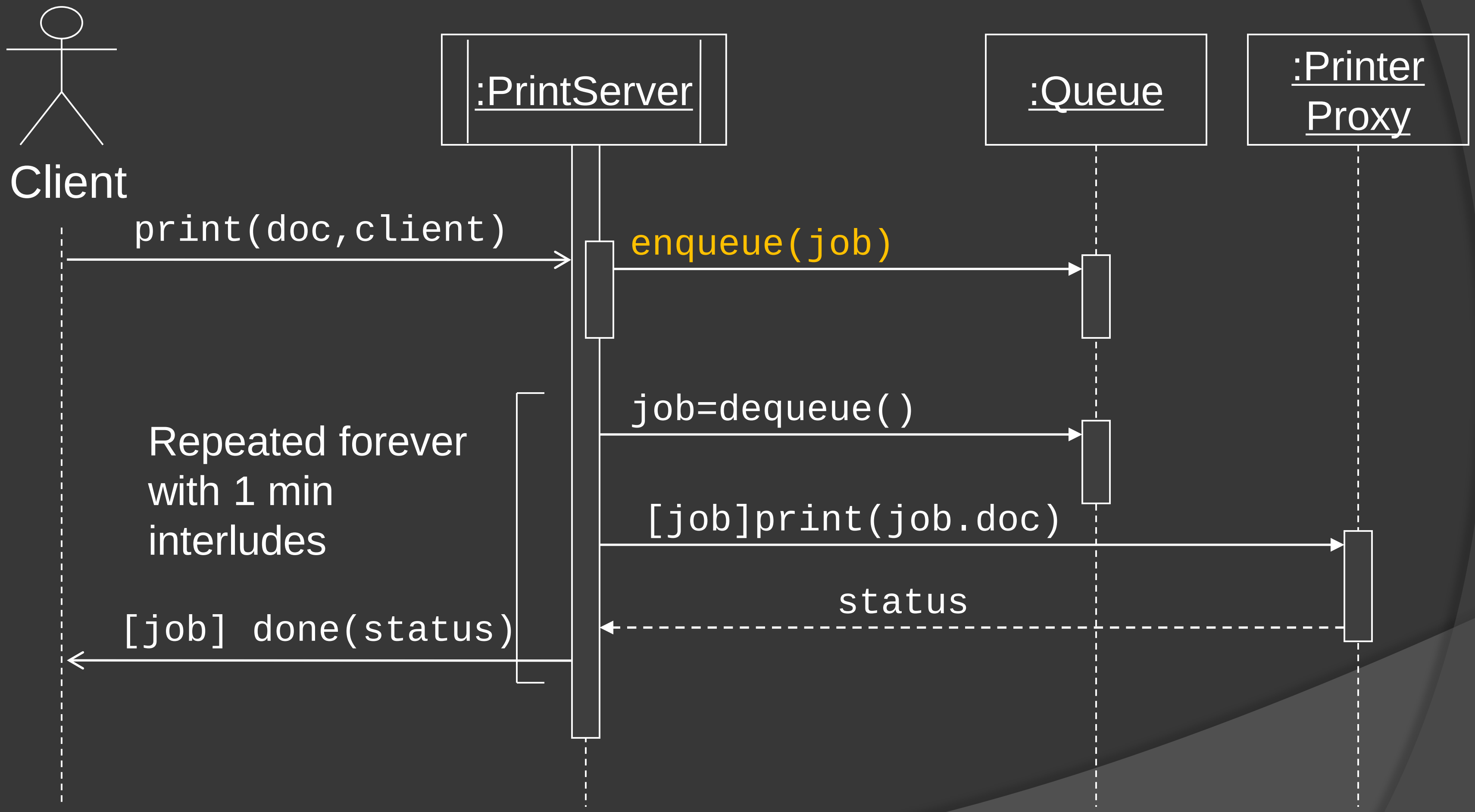

Client
:PrintServer
:Queue
:Printer Proxy
print(doc,client)
enqueue(job)
job=dequeue()
[job]print(job.doc)
status
[job] done(status)
Repeated forever with 1 min interludes

Examples

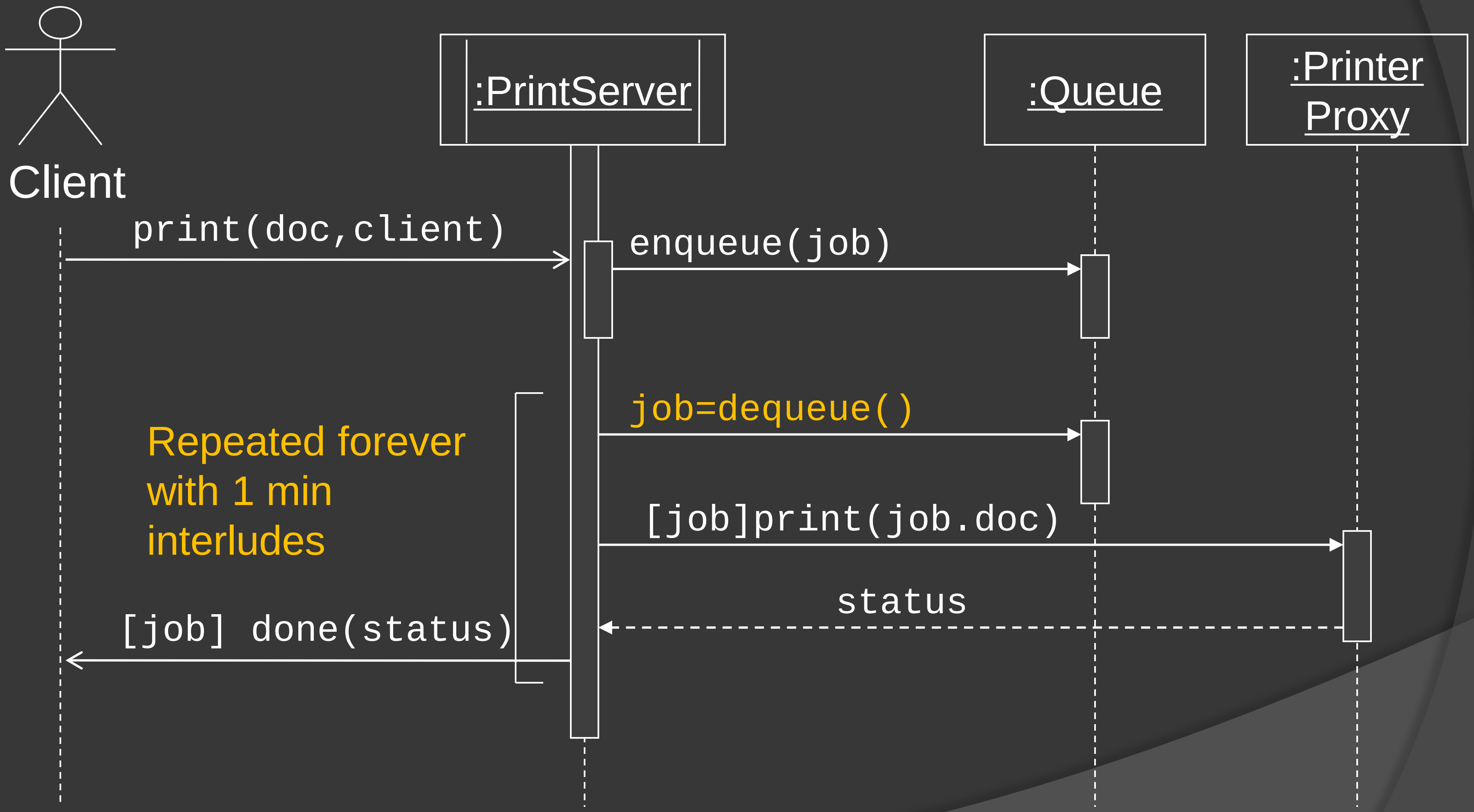


Client
:PrintServer
:Queue
:Printer Proxy
print(doc,client)
enqueue(job)
job=dequeue()
[job]print(job.doc)
status
[job] done(status)
Repeated forever with 1 min interludes

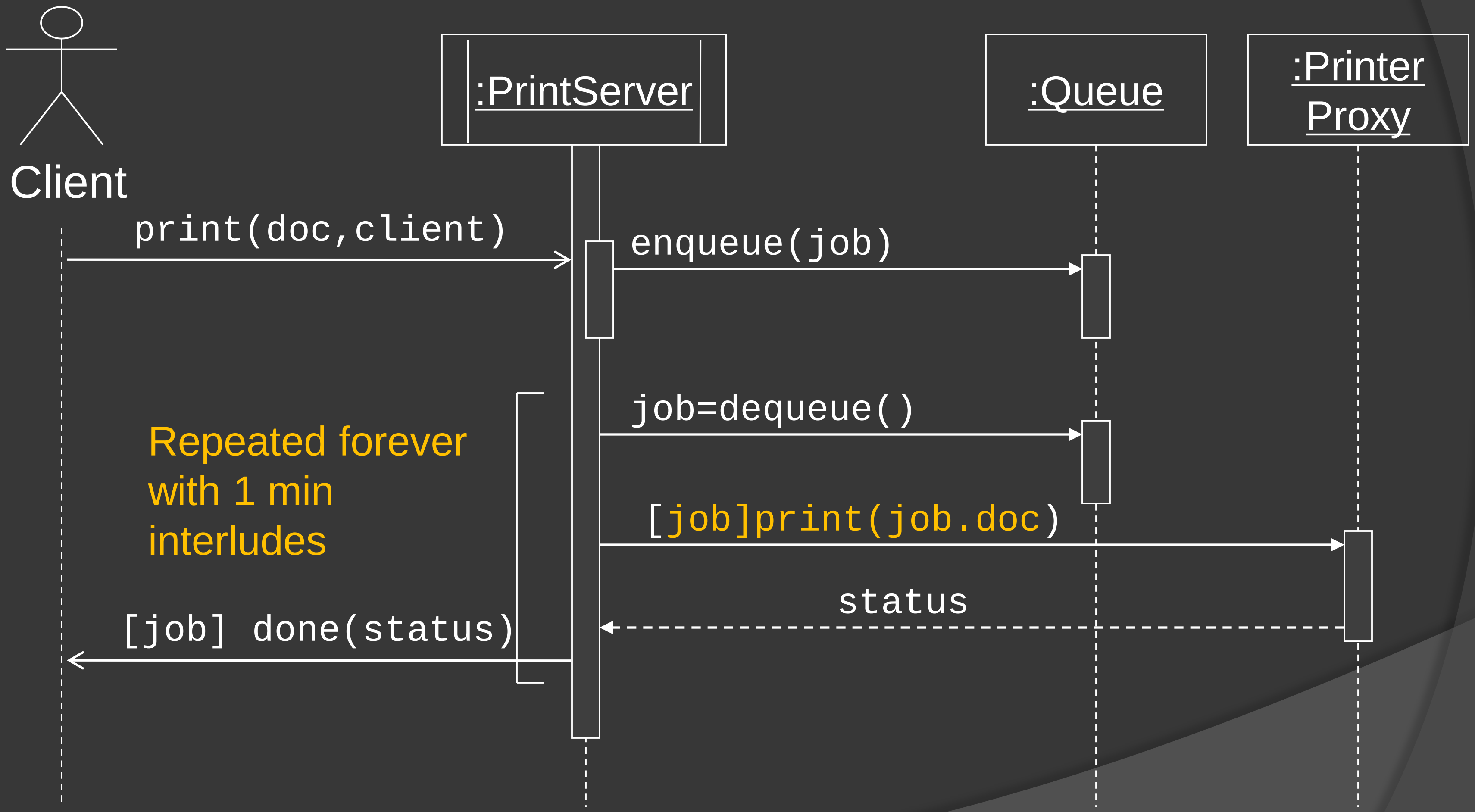
Examples


Client
:PrintServer
:Queue
:Printer Proxy
print(doc,client)
enqueue(job)
job=dequeue()
[job]print(job.doc)
status
[job] done(status)
Repeated forever with 1 min interludes

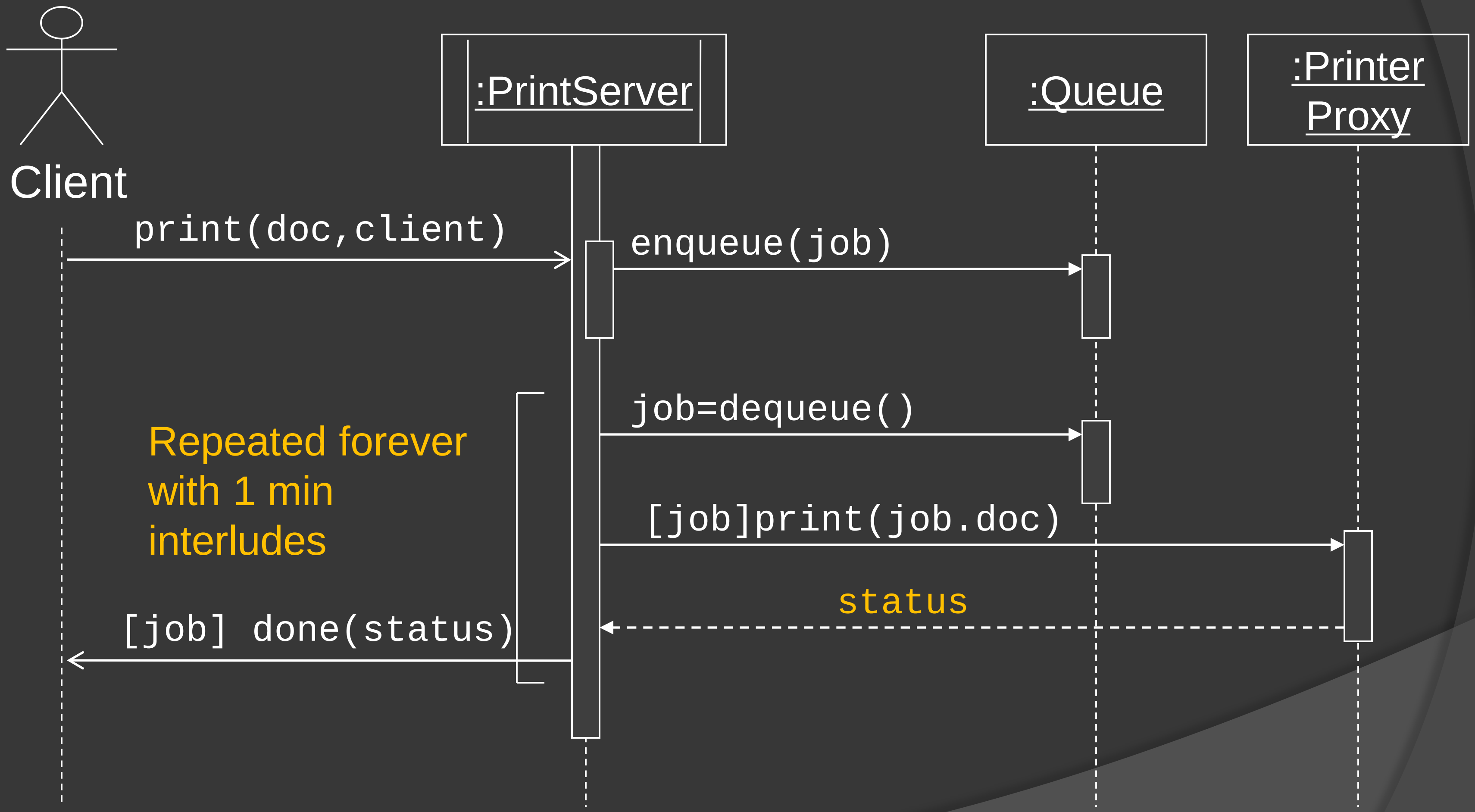
Examples


Client
:PrintServer
:Queue
:Printer Proxy
print(doc,client)
enqueue(job)
job=dequeue()
[job]print(job.doc)
status
[job] done(status)
Repeated forever with 1 min interludes

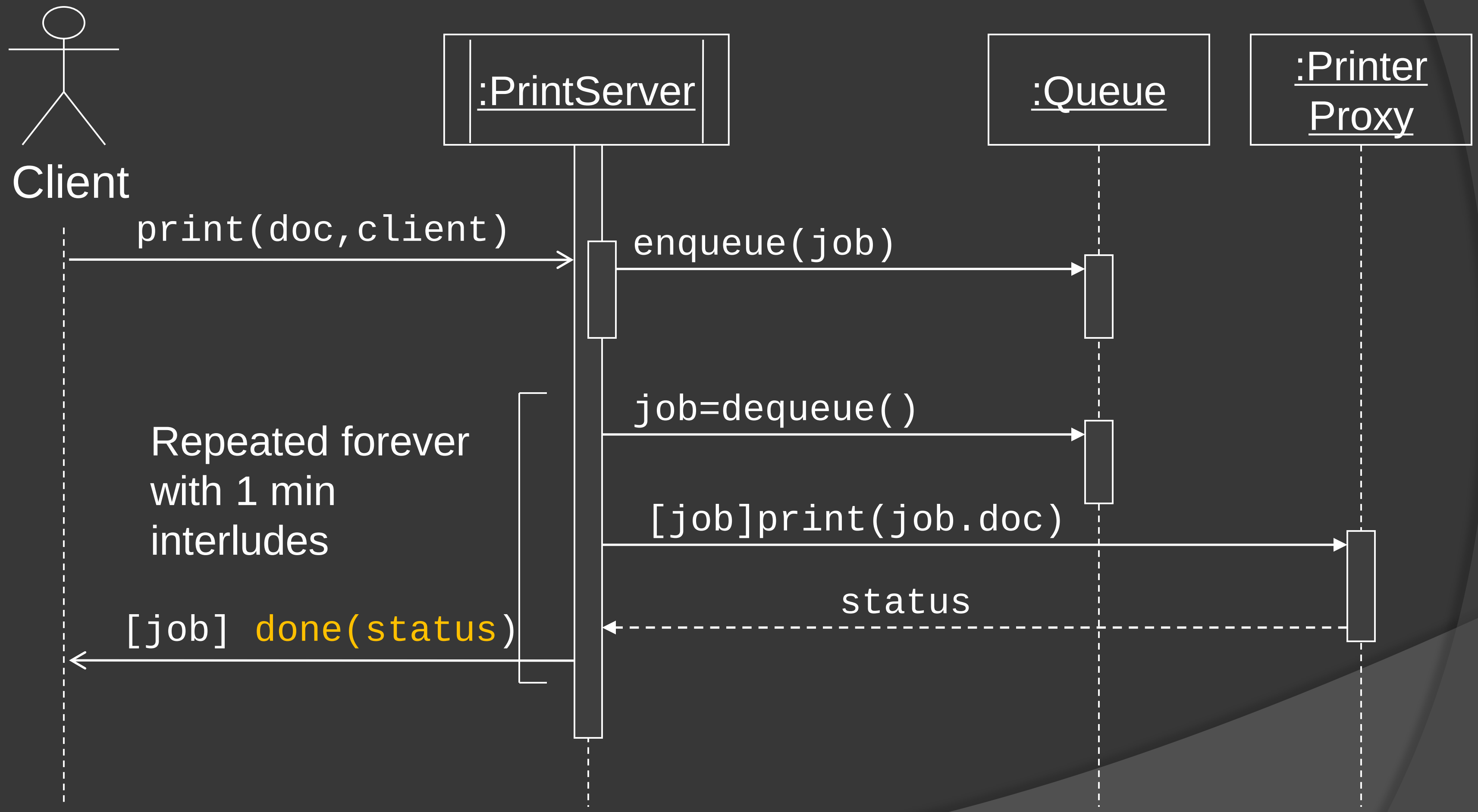
Examples


Client
:PrintServer
:Queue
:Printer Proxy
print(doc,client)
enqueue(job)
job=dequeue()
Repeated forever with 1 min interludes
[job]print(job.doc)
status
[job] done(status)

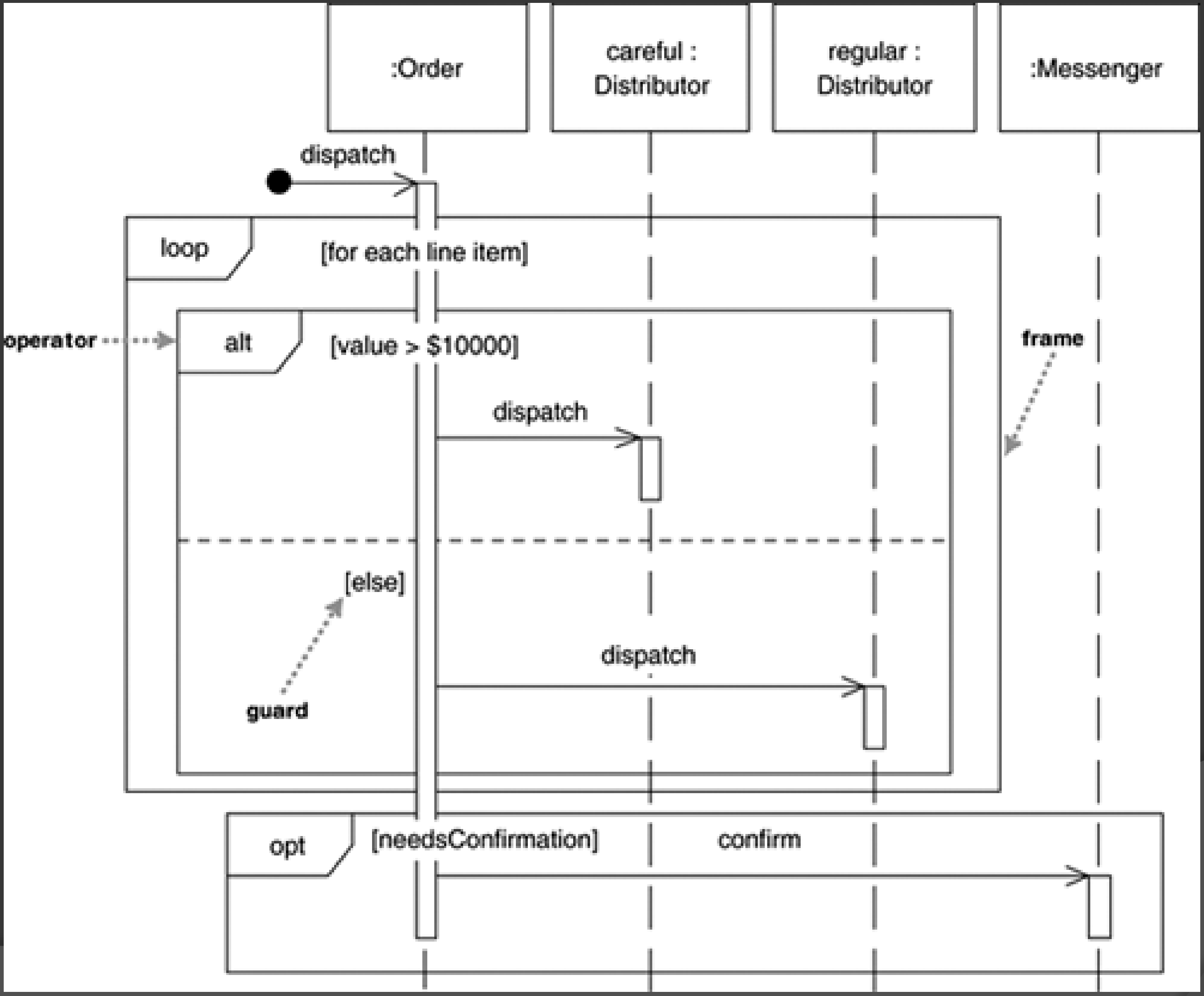
Examples


Client
:PrintServer
:Queue
:Printer Proxy
print(doc,client)
enqueue(job)
job=dequeue()
[job]print(job.doc)
status
[job] done(status)
Repeated forever with 1 min interludes

Examples

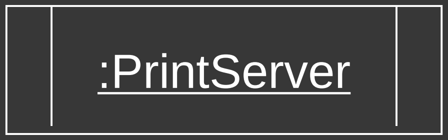

Client
:PrintServer
:Queue
:Printer Proxy
print(doc,client)
enqueue(job)
job=dequeue()
[job]print(job.doc)
status
[job] done(status)
Repeated forever with 1 min interludes

Loops, conditionals, …



Summary

- Time on y-axis, roles on x-axis
- Activation bars represent executions of procedures
- Active objects have bars
- Solid arrowheads: synchronous
- Stick arrowheads: asynchronous
- Dashed message lines: return



:PrintServer

CRC Cards

- CRC card = Class Responsibility Collaborator Card
- Beck & Cunningham

CRC Cards

- Help explore objects
- Provide an ’easy' introduction
- Starting point of many methodologies
- Used in industry
- Widely used in teaching

CRC Cards –format

- Index cards
- Post-It notes
- Walls/Whiteboards/Desks
- String and Blu-tack

Video on CRC card method

- https://www.youtube.com/watch?v=Bxgn6qJ-bYY

CRC Cards

	Class_Name	
Responsibility1		Collaboration1
Responsibility2		Responsibility2
…		…

- Responsibility = what class does or knows
- Collaborators = which classes help it perform the responsibility

Finding classes:

- Read specification
- Work through requirements, highlighting nouns and noun phrases to give candidate classes.
- Work through candidates, deciding likely classes and rejecting unlikely.

Read specification

- If you don't have one, WRITE your own.
- The specification should:
  - describe the goals of the design
  - discuss the things the system should do
- i.e. desired responses to expected inputs

Highlight noun phrases

- convert plurals to singular
- discard obvious nonsense classes (but keep all rest)
- remove synonyms (but keep BEST descriptor)
- beware of adjectives (they can be irrelevant, but can mean a whole new class exists)
- beware hidden nouns e.g. **passive voice** "the thing is activated" = "SOMETHING activates the thing"

Candidate classes

- physical objects
  - e.g. printer, switch
- cohesive entities
  - e.g. file, window
- categories of classes
  - (may become abstract superclasses)
- interfaces both to user and to other programs
- attribute values (NOT attributes) e.g. "circle has radius in real numbers" : circle and real are classes; radius is not.

Finding classes - problems

- Warnings
  - adjectives
  - passive voice
- Reject:
  - attributes
  - nouns that are really verbs
  - objects outside system

Identifying responsibilities

- Responsibilities are concerned with:
  - the maintenance of knowledge
  - the actions the object can perform
- Technique:
  1. Highlight verbs/phrases in requirements
  2. Do walkthroughs
  3. Spread intelligence
  4. Keep behaviour and knowledge close

What are responsibilities?

- They contain two key items:
  - the **knowledge** that the object maintains
  - the **actions** the object can perform
- They say WHAT gets done, not HOW its done

Identifying Responsibilities

- Read requirements & highlight:
  - verbs
  - information (that some object must maintain)
- Check these are actions that a system object must perform
- Try a walkthrough
  - try anthropomorphism/personification
- Check that all your classes are doing something useful

Assigning responsibilities

- distribute system intelligence
- state responsibilities as generally as possible
- keep behaviour with related information (if any)
- keep information about one thing in one place
- share responsibilities among related objects

Look at relationships between classes

Taking classes from within your system, see if there are examples of:

- **is-kind-of**
  - maybe superclass should have responsibility?
- **is-analogous-to**
  - if have similar responsibilities, perhaps should be common superclass with it?
- **is-part-of**
  - therefore clarify responsibilities between parts of an aggregate class

Difficulties

- Missing classes
  - perhaps worth encapsulating unassigned responsibilities to a new class?
- Uncertain Assignment of Responsibilities
  - i.e. maybe one responsibility could go to two different classes
  - solve by walkthrough?

Collaborations

- A collaboration:
  - one class (a client) needs another one (a server) in order to perform its own responsibilities.
  - NB this is a one-way relationship
- Each responsibility may have:
  - no collaborations
  - one collaboration
  - many collaborations

Finding collaborations

- Using CRC cards, work through ALL responsibilities & identify collaborators;.
- For each responsibility ask:
  - can the class do it alone?
  - if not:
    - what knowledge does it need?
    - what processing does it need?
    - which classes have what it needs?

Finding collaborations

- For each CRC card class ask:
  - what does it do or know?
  - what classes need this service or knowledge?
  - are those classes collaborating with this class?
- Confirm by looking at classes and ensure if a class does/knows something, it is being used

Examine relationships

- Collaboration is **strongly** indicated by:
  - **has-knowledge-of** e.g. a car needs to know the speed limit, which it gets from the sign on the side of the road or ...
  - **depends-on (changes-with)** e.g. pressing accelerator increases speed of wheels and decreases petrol level (but beware INDIRECT collaborations)
  - **(composite) is-part-of** e.g. to turn a car, the car will need to send messages to its steering wheel, wheels

Examine relationships

- Collaboration is **not** necessarily indicated by **container (is-part-of)** aggregation
- **For example** an array holds a group of elements but doesn’t usually collaborate with them as it doesn’t need to know their values or use any of their methods

An example use

- Paul Gestwicki, citizen-journalist example